# Deep Dive into Model-Driven Design

**Website**: https://www.avanscoperta.it/en/training/deep-dive-into-model-driven-design-workshop/

**Trainer: Bruno Boucard** and **Kenny Baas-Schwegler**

## Refactoring toward deeper insight

Refactoring toward deeper insight represents an advanced approach to refining software development processes, emphasizing not just the technical aspects of coding but also the incorporation of domain knowledge and collaboration with domain experts. This strategy transcends the traditional boundaries of code optimization, advocating for a holistic integration of understanding and technique to achieve superior software models. Let's distill the essence of this approach into actionable insights:

### 1. Live in the Domain

To effectively refactor for deeper insight, developers must immerse themselves in the domain of the software they're creating. This means understanding the business or application area superficially and deeply, getting to the heart of what the software aims to solve or improve. This immersion allows developers to anticipate needs, identify missing elements, and create more intuitive and practical solutions.

### 2. Adopt Diverse Perspectives

Refactoring isn't just about cleaning up code; it's about rethinking it from the ground up. This requires looking at the software from multiple angles and considering how different design choices impact its functionality, maintainability, and relevance to the domain. Encouraging diverse perspectives through personal reflection or collaboration fosters creative problem-solving and leads to more innovative and practical designs.

### 3. Engage with Domain Experts Continuously

Continuous dialogue with domain experts ensures that the evolving software model remains aligned with domain realities and leverages the deep knowledge of those who are experts in the field. This collaboration bridges the gap between technical implementation and practical application, ensuring the software accurately represents and serves its intended domain.

## Implementation Strategies

- **Initiation**: Recognize when traditional refactoring isn't enough and a deeper understanding of the domain is needed to address underlying complexities or shortcomings in the model.
- **Exploration Teams**: Form temporary, agile teams focused on specific design challenges, incorporating diverse expertise and domain knowledge to brainstorm and prototype solutions.
- **Prior Art**: Utilize existing knowledge, including analysis and design patterns, to inform the refactoring process, ensuring not to reinvent the wheel unnecessarily but to adapt and innovate as needed.

- **A Design for Developers**: Create supple designs that facilitate developers' understanding, modification, and extension. Recognize that software design serves not only the end-user but also those who maintain and evolve it.
- **Timing**: Act proactively to refactor for deeper insight, avoiding the trap of delaying until a comprehensive justification is apparent, as this may lead to missed opportunities and increased technical debt.
- **Crisis as Opportunity**: Leverage moments of crisis or deep realization as catalysts for significant breakthroughs in understanding and model refinement, embracing the punctuated equilibrium model of evolution in software development.

## Conclusion

This approach requires a shift in mindset from seeing software development as purely a technical endeavor to viewing it as an ongoing, collaborative process of learning and adaptation within a specific domain. By prioritizing domain immersion, encouraging diverse viewpoints, and maintaining an open dialogue with domain experts, developers can create software that meets current needs and is adaptable to future challenges and opportunities.